Planning for Robots with Skills

Planning for Robots with Skills

Matthew Crosby,
Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
{M.Crosby,R.Petrick}@hw.ac.uk

Francesco Rovida,
Mikkel Rath Pedersen, Volker Krüger
Robotics, Vision, and Machine Intelligence Lab
Aalborg University
Copenhagen 2450, Denmark
{francesco,mrp,vok}@m-tech.aau.dk

Abstract

Modern industrial robotics is characterised by a need for flexibility in robot design, in order to minimise programming and development time when a robot's tasks must be changed. To address this problem, a recent approach has proposed that robots be equipped with a set of general, reoccurring operations called 'skills', e.g., picking, placing, or driving. This paper presents a method for automatically generating planning problems from existing skill definitions such that the resulting problems can be solved using off-the-shelf planning software, and the solutions can be used to control robot actions in the world. As a result, a robot can therefore perform new tasks simply by specifying the task's goals via a GUI. The approach is demonstrated on a set of common tasks in a simulated industrial environment and has also been tested successfully on a real-world robotic platform.

Introduction

Robot autonomy is becoming increasingly important in modern industrial robotics, where factory robots often possess low degrees of autonomous operation at the task level, with a relatively large proportion of time spent on robot programming, compared with the time the robots spend performing tasks. This has important consequences for the current trend towards flexible manufacturing which requires frequent changeovers to new products: when a changeover occurs, the robots must be reprogrammed for the new tasks.

Task-level programming provides one way of simplifying the robot control problem. In this paradigm, a human programmer specifies what the robot should do in terms of the high-level actions and objects involved in a task, rather than focusing on the low-level details of the robot or its operating space. Actions are abstracted in a way which hides the complexity of the lower layers from the programmer, allowing users to focus on the task itself. The result is a powerful way to speed up programming, even with complex robots.

One proposal for implementing such a programming framework is based on defining tasks as sequences of *skills*, where skills are identified as the re-occurring actions needed to execute standard operating procedures in a factory (e.g., operations like *pick 'object'* or *place at 'location'*) (Madsen et al. 2015; Pedersen et al. 2016). Embedded within the skill definitions are the sensing and motor operations, or *primitives*, that accomplish the goals of the skill, as well as a set of condition checks that are made before and after execution to ensure robustness. This methodology also provides a process for specifying high-level parameters for skills, while low-level parameters for the primitive operations are mostly inferred through autonomous reasoning by the robot.

Figure 1: A robot operating in a factory environment using the SkiROS system. The robot is executing a six-step plan to place two parts in the white kitting box it is carrying.

While skills have been shown to be a useful tool for human operators to programme robot tasks (Madsen et al. 2015), the goal of increased robot autonomy in the factory environment also relies on the robots *themselves* being able to automatically sequence skills to perform tasks. For instance, when a new skill is introduced to a robot, a skills expert must specify the skill in terms of its input parameters, how it should be executed using low-level primitives, the conditions that must hold of the world state, and the intended changes to the world model. Previous work (Pedersen and Krüger 2015) showed how skill definitions of this form enabled planning problems to be created by hand and used to drive robot actions.

This paper instead introduces techniques for automating the creation of planning domains from the robot's skills and world model, so that the entire process of robot control can itself be automated. In particular, this work focuses on how

a planning problem can be automatically generated from the skill definition itself and, given a world model and a set of goals, how a sequence of parameterised skills can be constructed to achieve these goals. This has important consequences for robot control: using this system, and provided the appropriate skills are implemented, only the goals of the task need to be specified for a robot to complete a new task. This process is demonstrated with a set of skills (e.g., drive, pick, and place) implemented in a skills framework called *SkiROS* (Rovida and Krüger 2015), for a simulated robot system designed for a real factory environment. This approach has also been tested in a factory setting using a real robot and the same set of skills (see Figure 1).

The rest of this paper is organised as follows. First, the related work is considered. Then, the system architecture is introduced including a description of the skills framework. The world model is then outlined, followed by a description of the task planner and the process for converting skills to PDDL. The paper concludes with a set of experiments performed in simulation that demonstrate the system functioning in a mock factory environment resembling the real-world environment for which this system has been implemented.

Related Work

During the last three decades, three main approaches to robot control have dominated the research community: reactive, deliberative, and hybrid control (Kortenkamp and Simmons 2008). Reactive systems rely on a set of concurrently running modules, called behaviours, which directly connect input sensors to particular output actuators (Arkin 1998; Brooks 1986). In contrast, deliberative systems employ a sense-plan-act paradigm, where reasoning plays a key role in an explicit planning process. Hybrid systems attempt to exploit the best of both worlds, through mixed architectures with a deliberative high level, a reactive low level, and a synchronisation mechanism in the middle that mediates between the two (Firby 1989). Most modern autonomous robots use a hybrid approach (Gat 1998; Ferrein and Lakemeyer 2008; Bensalem and Gallien 2009; Magnenat 2010), with researchers focused on finding appropriate interfaces between declarative high-level reasoning and procedural low-level control.

SkiROS (Skills-ROS) (Rovida and Krüger 2015), the skills architecture used in this paper, is a hybrid framework following concepts from model-driven software engineering (Vanthienen, Klotzbücher, and Bruyninckx 2014; Schlegel et al. 2015). SkiROS splits the robot programming process into several layers of abstraction, with two main goals: (i) provide a state-of-the-art architecture for autonomous robot control, and (ii) make high-level robot programming simple and accessible even to non-experts.

Knowledge representation also plays a fundamental role in cognitive robotic systems (Vernon, von Hofsten, and Fadiga 2010), especially with respect to defining world models formalised in an ontology. A prominent example of knowledge processing in robotics is the KnowRob system (Tenorth and Beetz 2012; 2013), which combines knowledge representation and reasoning methods for acquiring and grounding knowledge in physical systems. KnowRob uses a semantic library which facilitates loading and accessing ontologies represented in the Web Ontology Language (OWL). KnowRob uses the ontology to store semantic representations of the world scene in order to reason about object positions in space and time, along with models of the robot hardware and the robot skills. A similar approach is presented in (Björkelund et al. 2012; Stenmark and Malec 2013; Bjørkelund and Edstrom 2011) as part of the Rosetta project, which focuses on how skills should be modelled for industrial assembly tasks. A similar study in (Huckaby 2014) defines a precise taxonomy of skills. However, none of these projects integrate skills into a consistent framework.

Automated planning has also been used for autonomous robot control since the days of Shakey (Nilsson 1984). While early approaches largely separated symbolic planning from other forms of planning like geometric planning, it was recognised that solutions often benefited from a hybrid approach (Cambon, Alami, and Gravot 2009). Recently, robot task planning has become an active research area, with approaches taken from diverse areas such as sampling-based motion planning (Plaku and Hager 2010; Barry 2013), integration of symbolic planning with robot-level processes (Dornhege et al. 2009), and probabilistic back-chaining (Kaelbling and Lozano-Pérez 2013).

A typical approach to robot task planning is to evaluate symbolic actions in a forward manner, sampling geometric choices and backtracking on failure. For instance, (Cambon, Alami, and Gravot 2009) use a symbolic planner that follows several heuristics to guide a geometric search. Symbolic and geometric searches are interleaved, with backtracking in both layers, and probabilistic roadmaps created for all combinations of robot manipulators and objects to represent the search space. Approaches like (Eiter et al. 2006; Dornhege et al. 2009; Erdem et al. 2011; Gaschler et al. 2013) add robot-level functions to a symbolic planning problem through an interface that allows external processes to be invoked during high-level planning. Other approaches like (Srivastava et al. 2014) solve scenarios given symbolic explanations for all failures in the geometric search, which are fed back to the symbolic search. Kaelbling and Lozano-Pérez (2013) perform a hierarchical, back-chaining search, combining geometric abstractions at the robot level with belief space planning.

Other approaches that attempt to bridge the gap between high-level and low-level robotics actions include ROSco (Nguyen et al. 2013) and Smach (Bohren and Cousins 2010) which use Hierarchical Finite State Machines as opposed to the planning approach taken in this paper. Approaches that use planning include ROSPlan (Cashmore et al. 2015) and the work of Vaquero et al. (2015). The former requires manual definition of planning domains, while the latter uses a translation approach specific to their application domain. In contrast, SkiROS is designed so that the user can define and modify skills on the fly, and the planning domains built to use these skills will be automatically generated.

Skills and the SkiROS Architecture

This section introduces the system architecture and skills model used in this work. Developing a robot system is, at

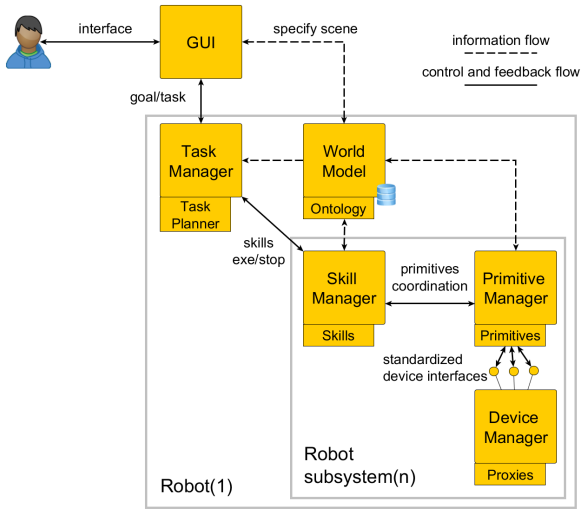

Figure 2: An overview of the SkiROS architecture. The robot presents an external interface from the task manager and the world model, accessed in this case by a GUI. The robot is composed of several subsystems, each one composed of a skill, primitive, and device manager. A skill coordinates the execution of several primitives to realise a world state change. The primitives implement atomic behaviours and interface to the hardware using standardised interfaces.

some level, a software engineering problem. However, robot architectures are distinguished from other software architectures by the special needs of robot systems. The most salient of these requirements, from a system design standpoint, is that robot systems must interact asynchronously and in real time with an uncertain, dynamic environment. At the same time, there is a need to define the tasks the robot can perform in a declarative way, in order to simplify task specification for end users. The skills model attempts to bridge this gap, with skills forming high-level building blocks that can be combined to solve complex tasks, yet containing all the necessary reasoning and control information to be executed by the robot in real time in a dynamic environment.

Skills Model

Robot skills like the ones in (Pedersen et al. 2016) can be thought of as general and robust software constructs that model self-contained, re-occurring operations that a robot might perform. Skills are intended to be designed such that they map easily to simple intuitive tasks. For example, a system might include calibration skills, manipulation skills for operations like picking and placing, as well as driving skills for mobile robots. Skills are implemented by experts to contain the necessary sensing and action operations for self-contained execution on the robot platform.

One benefit of a skills-based system is that non-experts can typically programme a robot task in a straightforward manner by selecting an appropriate skill sequence that results in the desired state changes to the robot's environment. This paper further removes the need for a non-expert user,

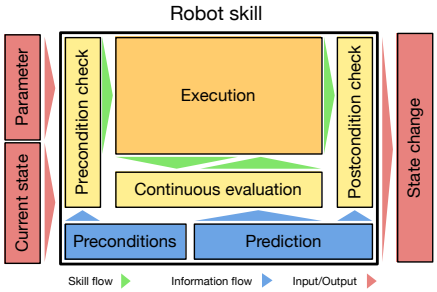

Figure 3: The conceptual model of a SkiROS skill.

and shows how skill sequences can be constructed using planning techniques in a completely automated way.

The skills framework used in this paper, SkiROS (Skills-ROS), is a Robot Operating System (ROS)[1] package implemented as an architecture with several layers of abstraction, as shown in Figure 2. The SkiROS architecture is designed to serve several tasks, including: (i) separating the bottom reactive layers from the top deliberative layers of the robot system, (ii) supporting hardware abstraction, and (iii) modularising robot programming to make it scalable.

The conceptual model of a robot skill is shown in Figure 3. A skill takes as input a set of parameters and a representation of the world state; it outputs a set of state changes. A skill contains both precondition and postcondition checks which monitor the environment, either through sensing or based on the world model. These checks allow the task layer to infer the likely causes of execution failures. For example, a precondition check for a pick skill might be that the item to be picked must be visible to a camera, and a postcondition check might be that the picked item must be in the gripper.

Skills Framework

The SkiROS framework is organised into four layers, each of which is represented by a manager. At the lowest layer is the *device manager*, which loads proxies (drivers which conform to a standard interface) and presents standard interfaces for similar devices (e.g., gripper, arm, camera, etc.). Standardised device interfaces extend the portability of all code, allowing drivers to be changed on the fly, for instance in the case of hardware changes like an updated end-effector. They also greatly simplify the switch between simulation and real-world execution.

The second layer contains the *primitive manager*, which contains motion primitives, software blocks that realise movement controlled with multi-sensor feedback, and services, software blocks that perform a generic computation. The modules are parameterised and loaded in the same way as a skill, but they don't have pre/postconditions and consist only of a *parameters specification* and *execution* part.

The third layer, the *skill manager*, loads skills and provides interfaces to the layer above. It also registers the robot subsystem on the world model, specifying the hardware,

[1] `http://www.ros.org/`

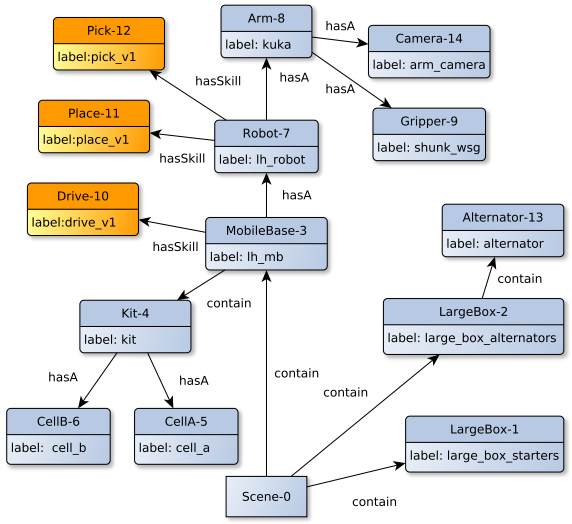

Figure 4: A simplified world model instance with physical (blue) and abstract (orange) objects. All physical objects are connected by a spatial relation in a scene graph structure.

available modules, and available skills. A skill's execution is usually implemented as a finite state machine which coordinates the execution of several parameterised primitives.

Finally, the fourth layer of the architecture is the *task manager* which monitors the presence of subsystems via the world model and acts as a general coordinator. The task manager is the interface for external systems, designed to be connected to a GUI or the manufacturing execution system (MES) of a factory. In this paper, the task manager is extended with an integrated task planner that takes as input a goal and snapshot of the world model and returns a sequence of skills to achieve the current task. The task planner, skills, primitives and proxies are imported as plug-ins using ROS.

World Model

In addition to skills, a key part of SkiROS is the *world model*, which acts as a knowledge integration framework. The world model is a vertical cross-layer component which links all layers together by gathering information from every subsystem at run time, allowing the modules to maintain a shared working memory, and storing the environment and skills information that are used to create the planning domain. In terms of the architecture, the world model can be read and modified from almost every part of the system.

The world state is partially predefined by a human operator in the ontology, partially abstracted from the robot by perception, and completed with the procedural knowledge embedded in the skills and primitives. Each skill manager in the system is responsible for keeping the world model updated with its subsystem information (e.g., hardware, available primitives, skill state, etc.). Similarly, each primitive and skill can extend the scene information with the results of robot operation or sensing. In special cases, the ontology can be extended automatically by the robot, to learn new concepts in a long-term memory (e.g., a new grasping pose).

Knowledge Integration

The core part of the robot's knowledge is organised into an OWL-DL ontology that can be efficiently embedded, edited, and extracted from the system. The SkiROS ontology is comprised of a set of classes C, a set of elements E, a set of relations R, and a set of properties P. Elements are individuals in a particular instance of the world model, for example a box or an alternator. Relations (OWL object properties) are binary relations that link two elements together, while properties (OWL data properties) are binary relations that link an element to a piece of typed data.

Every object in the world is represented in the scene as an *Element* class, which has the properties *type*, *id*, and *label*, along with a flexible list of other potential properties. The id links the Element to the scene, while the type and label categorise it in the ontology. The type is the most important property to this paper as it is used as the object's type in the planning translation. All other data associated with the Element are collected into the properties using a list of variants defined as *parameters*. For example, the Gripper element has the property *is_empty* which is initially set to `true` specifying that the gripper is empty. It is defined as a precondition check in the *Pick* skill to ensure that the gripper does not try to pick up an object while already holding one.

The set R of relations contains a special subset of spatial relations. Apart from the root *scene* element (which has no parent), each element in the world model has a spatial relation to exactly one parent element which ensures that the world model instantiation forms a tree (when only considering edges that represent spatial relations). This is particularly convenient for modelling the objects' spatial transformations. Example spatial relations from the current SkiROS ontology include `RobotAtLocation`, `Holding`, `Contains`, and `Carrying`. We write `RobotAtLocation(robot-1,largebox-1)` and say that `robot-1` is the subject and `largebox-1` is the object. It follows from the properties of this tree structure that an element cannot be both held by one element and contained in another at the same time, or that the robot cannot be in two locations at once.

Figure 4 shows an example instance of a world model. The tree formed by the spatial relations forms the scene graph, a data structure commonly used by modern computer games to arrange the logical and spatial representation of a graphical scene. In this structure, an object's pose is always defined with respect to the parent frame. The skills are connected to robot elements by the non-spatial relation *hasSkill*.

Skills are object-centric models that are parameterised with element types, while their instantiations are expected to link directly to elements of the appropriate type. A condition on a skill must specify either a relation or a property of an element in the world model. If a skill updates the world model by removing a spatial relation property, then it must also state the new subject related to that object as this cannot necessarily be inferred. Type information in skill relations must also be consistent with the world model.

Drive(MobileBase, Container) :

add: RobotAt(Container, MobileBase)

Pick(Gripper, Object, Container) :

pre: empty(Gripper)
pre: robotAt(Container, Robot)
pre: objectAt(Container, Object)
del: empty(Gripper)
add: contains(Gripper, Manipulatable)

Figure 5: Skill definitions in the SkiROS ontology.

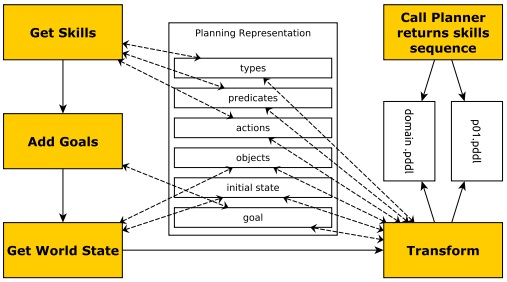

Figure 6: Overview of the task planning process and the creation of its internal planning representation.

Figure 5 shows the parameters, preconditions, and postconditions for the Drive and Pick skills, as defined in SkiROS. The preconditions (similarly, postconditions) are based on the expected and testable requirements of the state of the world prior to execution (similarly, after execution). Relations or properties for which postcondition checks are expected to be false become delete effects, and those expected to be true become add effects. Parameters are formed from the inputs needed for the skill's execution block.

Task Planner

The task planner has three main functions in SkiROS: it creates a PDDL representation of the skills, current state and goals; it calls an external planner to attempt to find a plan for the current goals; and, if a plan is found, it returns a sequence of skills to the task manager. The task planner creates a planning domain (and problem) written in PDDL 1.2 with only the `types` requirement. This means that the output is suitable for use with almost all modern planning systems. In what follows we use the standard definition of STRIPS-like planning actions (Fikes and Nilsson 1971) with *pre*, *add*, and *del* denoting the preconditions, add effects, and delete effects of an action, respectively.

An overview of the task planner is shown in Figure 6. This process is invoked (with the internal planning representation reset) every time the task manager requires a plan for completing the current set of goals. This is either triggered by an operator adding a goal via the SkiROS GUI, or by an external system (e.g., integrated with a factory MES) when SkiROS is deployed as part of a larger system. The central part of Figure 6 shows the task planner's planning library, which contains all the necessary structures to create a PDDL planning problem from the world state, skills, and goals. Specifically, this includes structures for types, predicates (both ground and unground), actions, and (typed) objects. In particular, the main body of a skill as shown in Figure 3 (surrounded by a black box) is not accessible to the task planner. Instead, the task planner uses the information accessible from the world model as shown in Figure 4.

Algorithm 1: Planning Domain Creation

Input : SkiROS World Model (wm), Goals ($goal$)
Output: Initial Planning Representation
`// Parse Skills`
1 **foreach** *Skill s : wm* **do**
2 types.addAllNewTypes(s)
3 predicates.addAllNewPredicates(s)
4 actions.addNewAction(s)
`// Add Goal State`
5 **foreach** *Goal g : goal* **do**
6 goals.add(g);
`// Parse World Model State`
7 **foreach** *Predicate p: predicates* **do**
8 initState.addAllTrueGroundings(p, wm)
9 objects.addAllNewObjects(p,wm)

Initial PDDL Creation

The process of creating the initial planning representation is given in Algorithm 1, which represents the left hand side of Figure 6 and involves three main steps. The first step is to parse the skills that exist in the world model. This involves adding all types and predicates that appear in the skills definitions to the planning library and also creating an action for each skill, which has a direct copy of the preconditions and effects. All relations, properties, and types that do not appear in a skill are therefore not included in the planning library.

The second step instantiates the goal. As goals are specified in the SkiROS GUI using the same predicates and objects in the world model that the skills use, they are simply added to the goal for the planning domain. If the goals contain a predicate or object that has not already been added to the planning library, then the planner returns an error message as no plan can exist given the defined skills.

Finally, the third step of the translation is to obtain the initial state of the planning problem from the current state of the world model. This process iterates over the predicates added in the previous step and queries the world model (through the SkiROS API) to find all ground instances of the predicates that are true. The objects contained in these predicates are added to the planning library as they are found.

The process of iterating through the skills and querying the world model to find true predicates may result in a planning library with less elements and types than in the world model. The omitted data can safely be ignored (and an error given for an incorrect goal) due to the following:

Lemma 1: *Any object with a type that does not appear in a skills definition can never appear in a solution plan.*

Algorithm 2: Planning Domain Refinement

Input : Initial Planning Representation
Output: Final Planning Representation

```
// Add Capabilities
```
foreach *Action a : actions* **do**
  predicates.add(can_a ?robot)
  a.pre.add(can_a ?robot)
  foreach *Robot r : hasSkill(a, r)* **do**
    initState.add(can_a r)

```
// Spatial Relation Constraints
```
foreach *Action* a **do**
  foreach *Spatial Relation* $S(o, s) \in a.add$ **do**
    if $\nexists S \in a.pre$ *AND* $\nexists S \in a.del$ **then**
      $s.params.add(x, s.type)$
      $s.pre.add(S(o, x))$
      $s.del.add(S(o, x))$
    else if $\nexists S \in a.pre$ **then**
      $s.pre.add(S(o, s))$
    else if $\nexists S \in a.del$ **then**
      $s.del.add(S(o, s))$

Proof Sketch: It is impossible to change the truth value of a predicate that does not appear in an action's effects, and the truth value of a predicate that does not appear in any action's preconditions can never be required for a change in state.

Domain Modification

The right hand side of Figure 6 deals with encoding the properties of the world model in the planning domain. The first part makes sure that skills are only usable by the correct elements, by querying the **hasSkill** relation from the world model. For each action, a new predicate `(can_a ?robot)` is added to the planning representation. This predicate is added as true in the initial state for each robot that can perform a particular skill and is invariant. An additional precondition `(can_a ?robot)` is added to each action to ensure that it can only be instantiated to the correct robots. If the `Robot` parameter is missing from the skill definition then this is added to the action parameters at this time.

The second part of the transformation step adds any preconditions and delete effects that are necessary to maintain the tree structure of the spatial relations in the world model. SkiROS contains methods for internally updating its world model so that it remains consistent, and these methods need to be included in the planning domain as they are not always made explicit in the skill definitions. For instance, referring to the skills in Figure 5, the Drive skill only contains a single predicate which specifies the new location of the robot. This is because the input for the execution block of the drive skill is only the goal location to which the robot has to move. The drive action must then be modified so that `robotAt` is true of only one grounding for the robot performing the drive skill, so the old instantiation must be found (it becomes a precondition) and added as a delete effect of the action.

The algorithm performs the steps in the previous example in a general manner that works for all spatial relations. It iterates over the skills in the planning library and checks each spatial relation in the add effects. If no corresponding spatial relation exists, in either the preconditions or delete effects of the action (i.e., no predicate with matching relation and subject as in the case of the drive skill), then a new predicate of the same spatial relation and the same object, but a new subject variable, is created and added to the preconditions and delete effects of the action. If a related spatial relation exists in just one of the preconditions and delete effects then it is added (with the same subject) to the other.

```
(:action drive
   :param (?R - Agent ?T - Location
     * ?preT - Location)
   :pre (and
     * (can_drive ?R)
     * (RobotAtLocation ?R ?preT))
   :eff (and
     * (not (RobotAtLocation ?R ?preT))
       (RobotAtLocation ?R ?T)))

(:action pick
   :param ( ?A - Arm ?C - Location ?G - Gripper
       ?O - Manipulatable ?R - Agent)
   :pre (and
       (EmptyHanded ?G)
       (RobotAtLocation ?R ?C)
       (ObjectAtLocation ?C ?O)
     * (can_pick ?R))
   :eff (and
       (not (EmptyHanded ?G))
     * (not (ObjectAtLocation ?C ?O))
       (Holding ?G ?O)))
```

Figure 7: The actions from Figure 5 after translation to PDDL. The asterisked lines are added by the translation.

Figure 7 shows the skills from Figure 5 after translation to PDDL. Note that in terms of implementation, the parameter added to the drive skill is removed when returning the parameterised skill to the task manager. The translation adds three new preconditions and two new delete effects over the two actions. The following lemma shows that these additions ensure the world model's tree structure is maintained:

Lemma 2: *Performing an action created by the task planner on a problem whose spatial relations form a tree will result in a state in which the spatial relations still form a tree.*

Proof Sketch: All that needs to be shown is that any deletion of a spatial relation property inserts it elsewhere with the same object (and therefore moves the whole subtree), and that every addition has a corresponding deletion. The former is a constraint on the skill definition. For the latter, every time a new spatial relation appears in the add effects then, by construction of the algorithm, a spatial relation with the same subject must appear in the delete effects. This spatial relation must match the only occurrence of that object in a spatial relation in the current state otherwise the action could not be performed as this must exist (again by construction) as a precondition to the action.

Once the translation is complete, the planning problem is written to domain and problem files in PDDL for use with an external planner. The planner's output (a sequence of instan-

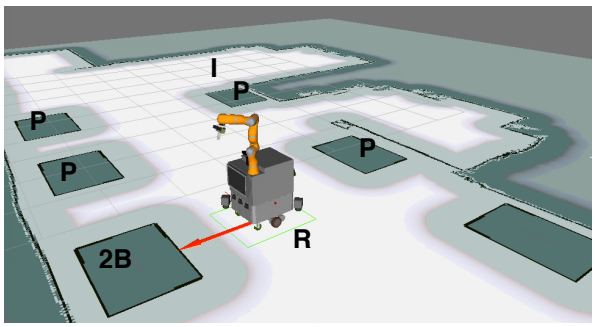

Figure 8: A visualisation of the simulated environment, showing an excerpt of the navigation map containing the robot idle location (I), a number of pallets (P), a pallet with two boxes of parts (2B), and the robot (R).

tiated actions) is parsed and converted back to parameterised skills to be sent to the task manager for robot execution.

Experiments

Figure 8 depicts the simulated environment used for testing.[2] In this setup, kits can contain six different parts: *engine support*, *thermal shield*, *compressor*, *tube*, *alternator*, and *starter*. Four parts (compressor, tube, alternator, and starter) are located in individual pallets; the two remaining parts (engine support and thermal shield) are located in smaller adjacent boxes on a single pallet.

In order to specify this setup in the robot's world model within SkiROS, the 2D poses (including orientation) of each pallet, and the parts contained within, need to be specified manually. The poses are defined in the world model coordinate system, and the transform between this frame and the robot navigation map frame is known. This information can be extracted automatically from the manufacturing execution system (MES) in real-world deployment; for obvious reasons, this is not possible for the simulated environment. The kit that is mounted on the robot is specified as a set of coordinate frames, with one parent frame defining the kit with respect to the robot, and the rest defining the individual compartments in the kit with respect to the kit itself. A part type is associated with each compartment in the kit.

The experiments used a simulated version of a mobile manipulator with an articulated robot arm mounted on a mobile platform. The robot arm was equipped with a 2-finger parallel gripper and an RGB-D camera mounted on the gripper. The execution of skills was simulated using the ROS interfaces employed by the real hardware drivers that were replaced. For example, the MoveIt arm motion planner (that outputs joint trajectories for an arm) and the navigation software (that outputs velocity commands to a mobile base) were not modified. Figure 4 shows a (slightly) simplified version of the spatial relations and components used in the experiment, with picking, placing, and driving skills.

[2]Visualisation was performed using rviz, a 3D tool for ROS (`http://wiki.ros.org/rviz`).

```
drive mobbase-2 loc-1 lbox-10
pick lbox-10 gripper-6 t_shield robot-3
drive mobbase-2 lbox-10 lbox-9
place grip-6 t_shield celld-19 kit-15 robot-3
pick lbox-9 gripper-6 starter robot-3
place grip-6 starter cellb-17 kit-15 robot-3
```

Figure 9: The plan found for the goal of placing two parts (thermal shield and starter) into a kit.

It is not possible, nor necessary, to simulate sensor information in this experiment. However, an inherent part of the skills is that they perform the necessary sensing operations to complete the skill. For this reason, a simple simulated object detection primitive is added, that places an object in the world model that is immediately in front of the robot.

This level of simulation makes it possible to visualise the robot system as it performs the skills, using the same skills that would be running on a real robot system. In terms of Figure 2, only the device layer and a single primitive (i.e., the object detection primitive) is simulated. Therefore the system uses, and is completely integrated in, a complete version of SkiROS, with the same skills as a real robot.

For the experiments, the FastDownward planner (Helmert 2006) was used, with A* search and the landmark-cut heuristic. Since the planning problems created by the translation process did not test the limits of the external planner, and are solved in less than a second (including world model querying, extraction, and translation), there was no benefit in comparing different planners. Instead, any state-of-the-art planner that supports the required features could be used.

Results

The first experiment tested a two skill setup in which only the robotic arm and the pick and place skills were used. The robot was placed in front of a pallet with two smaller boxes containing thermal shields and engine supports. The system was tested with the goal that one of each of the two different types of parts must be placed in the robot's kit. The extraction of the planning domain, and the planning itself, was completed in 0.6s, resulting in a plan with four skills that was successfully executed in 78s.[3] The experiment was then rerun with the goal specifying parts that were not in the vicinity of the robot. In this case, the task planner correctly returned that no plan could be found.

The second experiment introduced a second robot subsystem, the mobile base (which carries the robotic arm) and its associated drive skill. In this case, the task planner automatically included the drive skill in its planning domain. With this addition, a plan could be found for the previously unsolvable problem in which the parts are inaccessible without the ability to drive between locations. When the goal was specified to build a complete kit with six parts, with the robot finishing at an idle location, the PDDL extraction and planning took 0.9s. The resulting plan (with 18 skills) executed

[3]Planning, motion planning, simulation, SkiROS, and visualisation ran on a 2011 laptop with an i7@2.7GHz processor.

correctly in 371s. Figure 9 shows the plan for the goal of filling a kit with two parts (a thermal shield and a starter).

Overall, these experiments demonstrated that the task planning process is able to work as an integrated component in the SkiROS system and that the process is robust enough to find correct plans when different subsets of the currently implemented skills are enabled. The experiments also showed that the planning time is not a significant bottleneck (less than one second in all cases), especially when compared to execution time for these types of tasks.

Conclusions and Future Work

This paper presented a fully implemented software framework for deploying autonomous robot systems in an industrial setting. The system uses a skills model, called SkiROS, to bridge the gap between low-level robot control and high-level planning. Skills are declared explicitly and passed to a task planner which automatically generates the PDDL planning domain. The resulting system was shown to operate successfully in a simulated factory environment and has also been tested in a real-world factory setting. From an end-user perspective, the robot is programmed to perform new tasks by specifying goal conditions; new skills are added by specifying constraints on the world model with no explicit knowledge of planning required.

Work is progressing to test more skill implementations and further explore the relationship between skills and planning. Failure handling will be improved by extending the interaction of the planner with the task manager, to allow for replanning in the case of unsatisfied pre/postconditions and execution failures. To optimise cycle time, the assumption of sequential skill execution will be relaxed, allowing parallel skill execution from temporal plans.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant no. 610917 (STAMINA, `stamina-robot.eu`).

References

Arkin, R. C. 1998. *Behavior-based Robotics*. Cambridge, MA, USA: MIT Press, 1st edition.

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, MIT, USA.

Bensalem, S., and Gallien, M. 2009. Toward a more dependable software architecture for autonomous robots. *IEEE Robotics and Automation Magazine* 1–11.

Bjørkelund, A., and Edstrom, L. 2011. On the integration of skilled robot motions for productivity in manufacturing. In *IEEE International Symposium on Assembly in Manufacturing*.

Björkelund, A.; Malec, J.; Nilsson, K.; Nugues, P.; and Bruyninckx, H. 2012. Knowledge for Intelligent Industrial Robots. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Bohren, J., and Cousins, S. 2010. The smach high-level executive [ros news]. *Robotics & Automation Magazine, IEEE* 17(4):18–20.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *Journal of Artificial Intelligence Research* 2(1):14–23.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28(1):104–126.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the robot operating system. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 1–6.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications*, 273–287.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11):980–991.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. Dissertation, Yale University, USA.

Gaschler, A.; Petrick, R. P. A.; Giuliani, M.; Rickert, M.; and Knoll, A. 2013. KVP: A knowledge of volumes approach to robot task planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, 202–208.

Gat, E. 1998. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Huckaby, J. 2014. *Knowledge Transfer in Robot Manipulation Tasks*. PhD thesis, Georgia Institute of Technology, USA.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Kortenkamp, D., and Simmons, R. 2008. Robotic systems architectures and programming. In *Springer Handbook of Robotics*. Springer. 187–206.

Madsen, O.; Bøgh, S.; Schou, C.; Andersen, R. S.; Damgaard, J. S.; Pedersen, M. R.; and Krüger, V. 2015. In-

tegration of mobile manipulators in an industrial production. *Industrial Robot: An International Journal* 42(1):11–18.

Magnenat, S. 2010. *Software integration in mobile robotics, a science to scale up machine intelligence*. PhD thesis, École polytechnique fédérale de Lausanne, Switzerland.

Nguyen, H.; Ciocarlie, M.; Hsiao, K.; and Kemp, C. C. 2013. Ros commander (rosco): Behavior creation for home robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 467–474. IEEE.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Pedersen, M., and Krüger, V. 2015. Automated planning of industrial logistics on a skill-equipped robot. In *Workshop on Task Planning for Intelligent Robots in Service and Manufacturing at IROS 2015*.

Pedersen, M. R.; Nalpantidis, L.; Andersen, R. S.; Schou, C.; Bøgh, S.; Krüger, V.; and Madsen, O. 2016. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing* 37:282–291.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 5002–5008.

Rovida, F., and Krüger, V. 2015. Design and development of a software architecture for autonomous mobile manipulators in industrial environments. In *IEEE International Conference on Industrial Technology (ICIT)*.

Schlegel, C.; Lotz, A.; Lutz, M.; Stampfer, D.; and Vicente-Chicote, C. 2015. Model-Driven Software Systems Engineering in Robotics: Covering the Complete Life-Cycle of a Robot. *it - Information Technology* 57(2):85–98.

Srivastava, S.; Fang, E.; Lorenzo, R.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Stenmark, M., and Malec, J. 2013. Knowledge-based industrial robotics. *Scandinavian Conference on Artificial Intelligence*.

Tenorth, M., and Beetz, M. a. 2012. Knowledge Processing for Autonomous Robot Control. *Proceedings of the AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research* 32(5):566–590.

Vanthienen, D.; Klotzbücher, M.; and Bruyninckx, H. 2014. The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming. *Journal of Software Engineering for Robotics* 5(1):17–35.

Vaquero, T.; Mohamed, S. C.; Nejat, G.; and Beck, J. C. 2015. The implementation of a planning and scheduling architecture for multiple robots assisting multiple users in a retirement home setting. In *AAAI Workshop on Artificial Intelligence Applied to Assistive Technologies and Smart Environments*.

Vernon, D.; von Hofsten, C.; and Fadiga, L. 2010. *A Roadmap for Cognitive Development in Humanoid Robots*. Springer.